CFD_1

August 27, 2024

Let us start with the 1-D linear convection equation given as

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

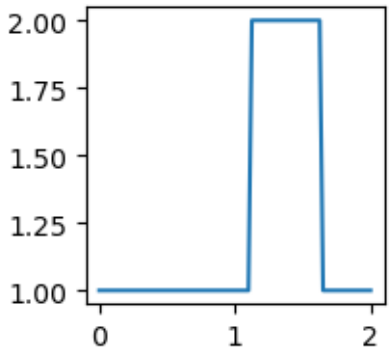
This equation represents the propagation of that initial wave with speed c. With initial condition $u(x,0) = u_0(x)$, exact solution of the equation is $u(x,t) = u_0(x - ct)$.

We can solve the equation with discretized

$$u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import time, sys
```

[2]:
```
nx = 81
dx = 2 / (nx-1)
nt = 25    #nt is the number of timesteps
dt = .025  # timestep covers
c = 1      # wavespeed of c = 1
u = np.ones(nx)
u[int(.5 / dx):int(1 / dx + 1)] = 2
un = np.ones(nx)

for n in range(nt):
    un = u.copy()
    for i in range(1, nx):
        u[i] = un[i] - c * dt / dx * (un[i] - un[i-1])
plt.figure(figsize=(2,2))
plt.plot(np.linspace(0, 2, nx), u);
```



The non-linear convection equation is given as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

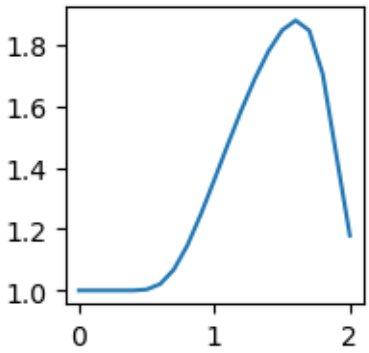
$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n \frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

Therefore we have the unknown term, u_i^{n+1}, as:

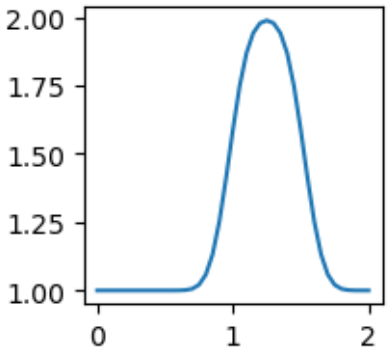
$$u_i^{n+1} = u_i^n - u_i^n \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

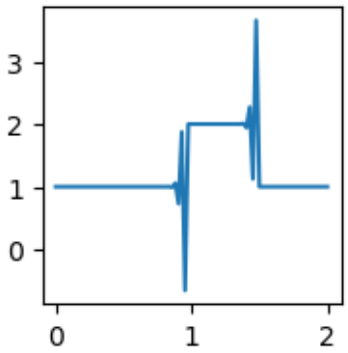
[3]:
```
nx = 21
dx = 2 / (nx - 1)
nt = 20
dt = .025

u = np.ones(nx)
u[int(.5 / dx) : int(1 / dx + 1)] = 2

un = np.ones(nx)

for n in range(nt):
    un = u.copy()
    for i in range(1, nx):
            u[i] = un[i] - un[i-1] * dt / dx * (un[i] - un[i-1])
plt.figure(figsize=(2,2))
plt.plot(np.linspace(0, 2, nx), u)
```

[3]:
```
[<matplotlib.lines.Line2D at 0x115400880>]
```



[4]:
```
def linearconv(nx):
    dx = 2 / (nx - 1)
    nt = 20
    dt = .025
    c = 1

    u = np.ones(nx)
    u[int(.5/dx):int(1 / dx + 1)] = 2

    un = np.ones(nx)

    for n in range(nt):
        un = u.copy()
        for i in range(1, nx):
            u[i] = un[i] - c * dt / dx * (un[i] - un[i-1])
    plt.figure(figsize=(2,2))
    plt.plot(np.linspace(0, 2, nx), u);
linearconv(41)
linearconv(85)
```





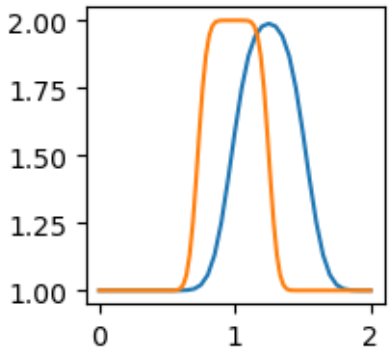
Over the time period Δt, the wave travelled a distance greater than `dx`. The length `dx` of each grid box is related to the number of total points `nx`, so stability can be enforced if the Δt step size is calculated with respect to the size of `dx`.

$$\sigma = \frac{u \Delta t}{\Delta x} \leq \sigma_{max}$$

where u is the speed of the wave; σ is called the Courant number and the value of σ_{max} that will ensure stability depends on the discretization used.

Corecting with CFL number we have:

[5]:
```
def linearconv(nx):
    dx = 2 / (nx - 1)
    nt = 20
    c = 1
    sigma = .5

    dt = sigma * dx

    u = np.ones(nx)
    u[int(.5/dx):int(1 / dx + 1)] = 2

    un = np.ones(nx)

    for n in range(nt):
        un = u.copy()
        for i in range(1, nx):
            u[i] = un[i] - c * dt / dx * (un[i] - un[i-1])

    plt.plot(np.linspace(0, 2, nx), u)
plt.figure(figsize=(2,2))
linearconv(41)
linearconv(85)
```



The one-dimensional diffusion equation is written as:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}$$

We can now write the discretized version of the diffusion equation in 1D:

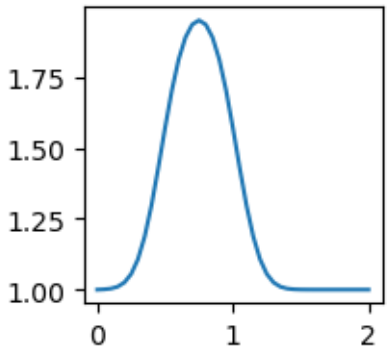
$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \nu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

On re-arranging the equation solving we get

$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

[6]:
```
nx = 41
dx = 2 / (nx - 1)
nt = 20    #the number of timesteps we want to calculate
nu = 0.3   #the value of viscosity
sigma = .2 #sigma is a parameter, we'll learn more about it later
dt = sigma * dx**2 / nu #dt is defined using sigma ... more later!


u = np.ones(nx)      #a numpy array with nx elements all equal to 1.
u[int(.5 / dx):int(1 / dx + 1)] = 2  #setting u = 2 between 0.5 and 1 as per␣
 ↪our I.C.s

un = np.ones(nx) #our placeholder array, un, to advance the solution in time

for n in range(nt):  #iterate through time
    un = u.copy() ##copy the existing values of u into un
    for i in range(1, nx - 1):
        u[i] = un[i] + nu * dt / dx**2 * (un[i+1] - 2 * un[i] + un[i-1])
plt.figure(figsize=(2,2))
```

```
plt.plot(np.linspace(0, 2, nx), u);
```



[ ]:

[ ]: